ABBOTTABAD UNIVERSITY OF SCIENCE & TECHNOLOGY

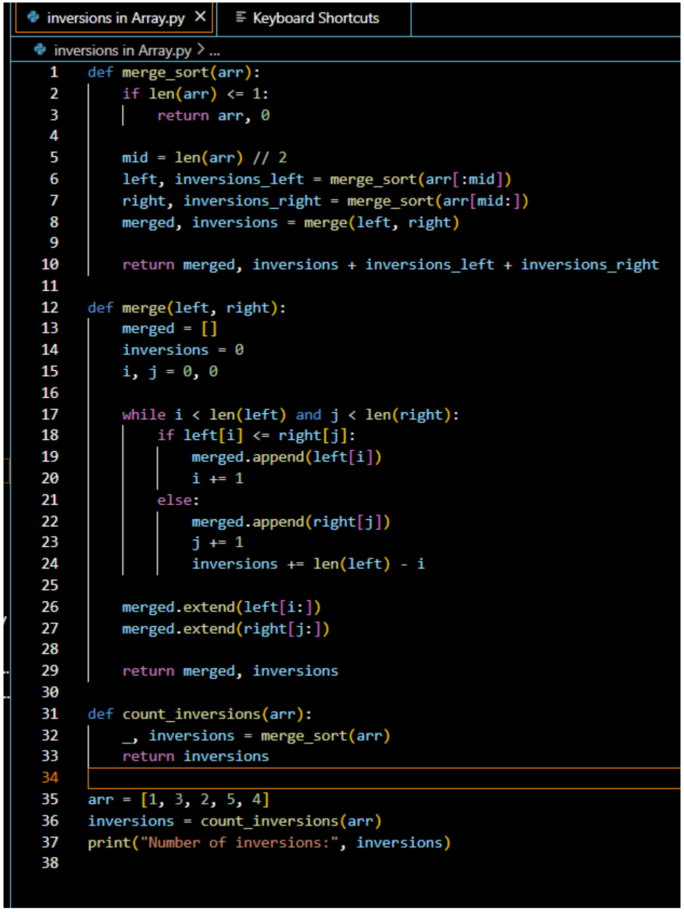
AUST

DEPARMENT OF SOFTWARE ENGINEERING

NAME	#	WAQAS AHMAD
ROLL_NO	#	12409
SEMESTER (SECTION)	#	3rd Semester (c)
SUBJECT	#	DSA (Lab : 04)
SUBMITTED TO	#	Mr.Abdul Ahad
DATE	#	6/11/2023

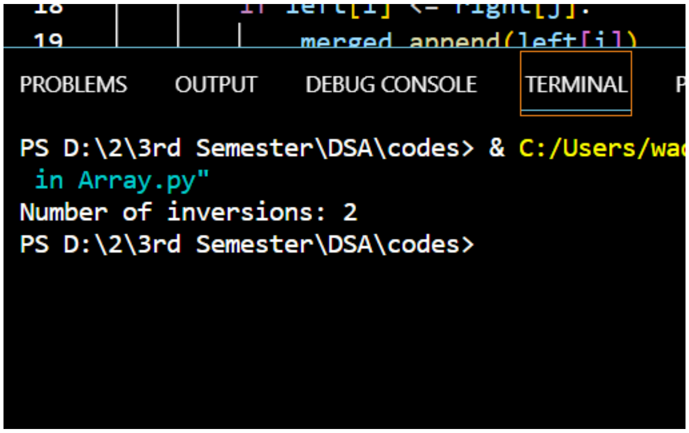
LAB : 04

1. Modify the merge sort algorithm to count the number of inversions in an array. An inversion is a pair of indices (i, j) such that i < j and arr[i] > arr[j].



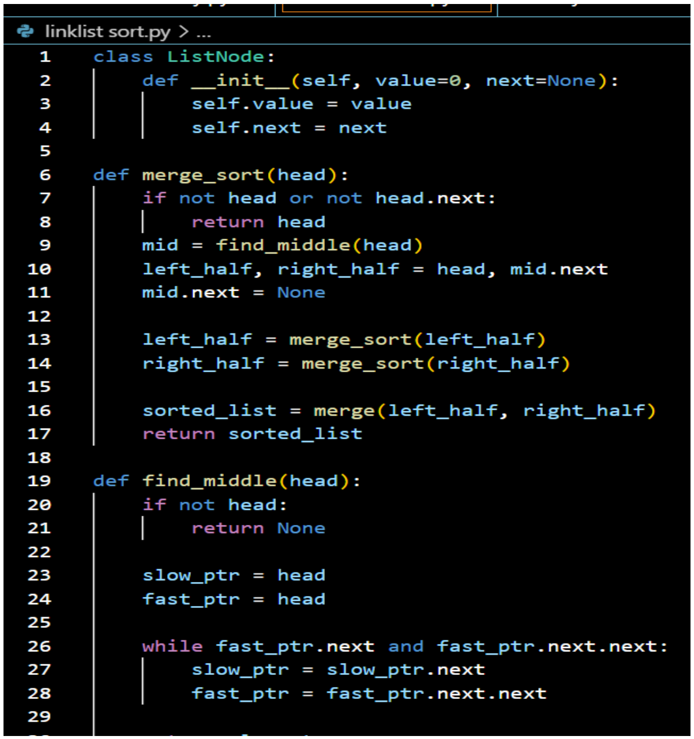
```python
def merge_sort(arr):
    if len(arr) <= 1:
        return arr, 0

    mid = len(arr) // 2
    left, inversions_left = merge_sort(arr[:mid])
    right, inversions_right = merge_sort(arr[mid:])
    merged, inversions = merge(left, right)

    return merged, inversions + inversions_left + inversions_right

def merge(left, right):
    merged = []
    inversions = 0
    i, j = 0, 0

    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
            inversions += len(left) - i

    merged.extend(left[i:])
    merged.extend(right[j:])

    return merged, inversions

def count_inversions(arr):
    _, inversions = merge_sort(arr)
    return inversions

arr = [1, 3, 2, 5, 4]
inversions = count_inversions(arr)
print("Number of inversions:", inversions)
```

OUTPUT:

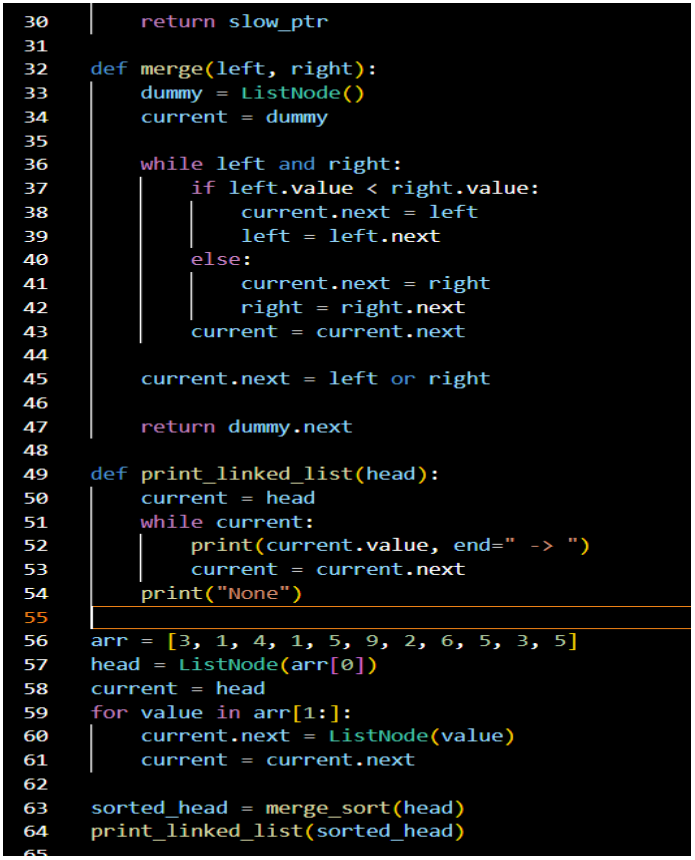


```
PS D:\2\3rd Semester\DSA\codes> & C:/Users/wa
 in Array.py"
Number of inversions: 2
PS D:\2\3rd Semester\DSA\codes>
```

2. Implement the merge sort algorithm for sorting linked lists instead of arrays. This exercise will require modifying the merge process.

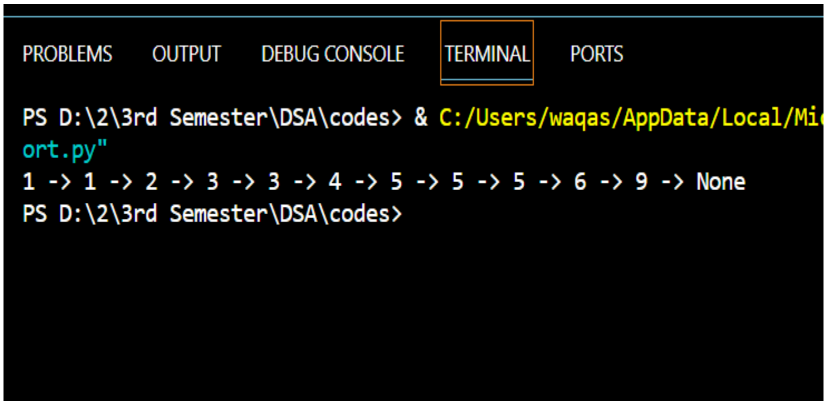


```python
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def merge_sort(head):
    if not head or not head.next:
        return head
    mid = find_middle(head)
    left_half, right_half = head, mid.next
    mid.next = None

    left_half = merge_sort(left_half)
    right_half = merge_sort(right_half)

    sorted_list = merge(left_half, right_half)
    return sorted_list

def find_middle(head):
    if not head:
        return None

    slow_ptr = head
    fast_ptr = head

    while fast_ptr.next and fast_ptr.next.next:
        slow_ptr = slow_ptr.next
        fast_ptr = fast_ptr.next.next

```



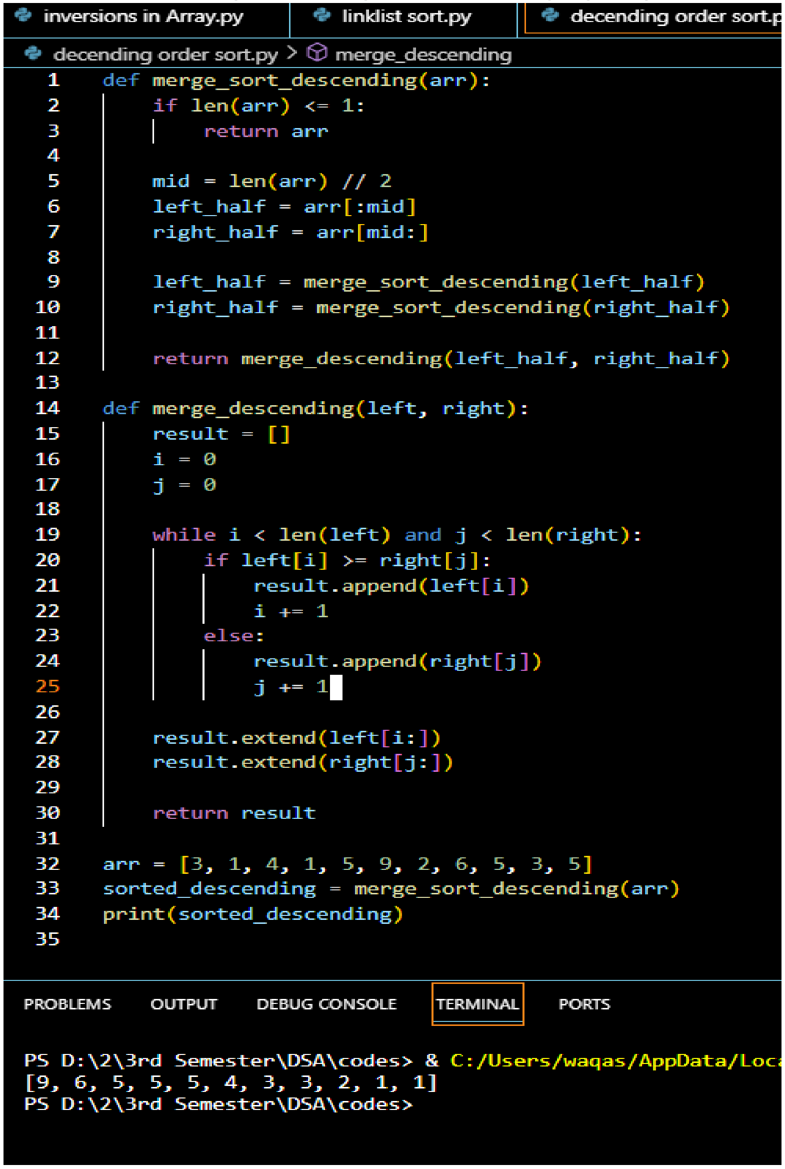
```python
    return slow_ptr

def merge(left, right):
    dummy = ListNode()
    current = dummy

    while left and right:
        if left.value < right.value:
            current.next = left
            left = left.next
        else:
            current.next = right
            right = right.next
        current = current.next

    current.next = left or right

    return dummy.next

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

arr = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
head = ListNode(arr[0])
current = head
for value in arr[1:]:
    current.next = ListNode(value)
    current = current.next

sorted_head = merge_sort(head)
print_linked_list(sorted_head)
```

Output:



```
PS D:\2\3rd Semester\DSA\codes> & C:/Users/waqas/AppData/Local/Mi
ort.py"
1 -> 1 -> 2 -> 3 -> 3 -> 4 -> 5 -> 5 -> 5 -> 6 -> 9 -> None
PS D:\2\3rd Semester\DSA\codes>
```

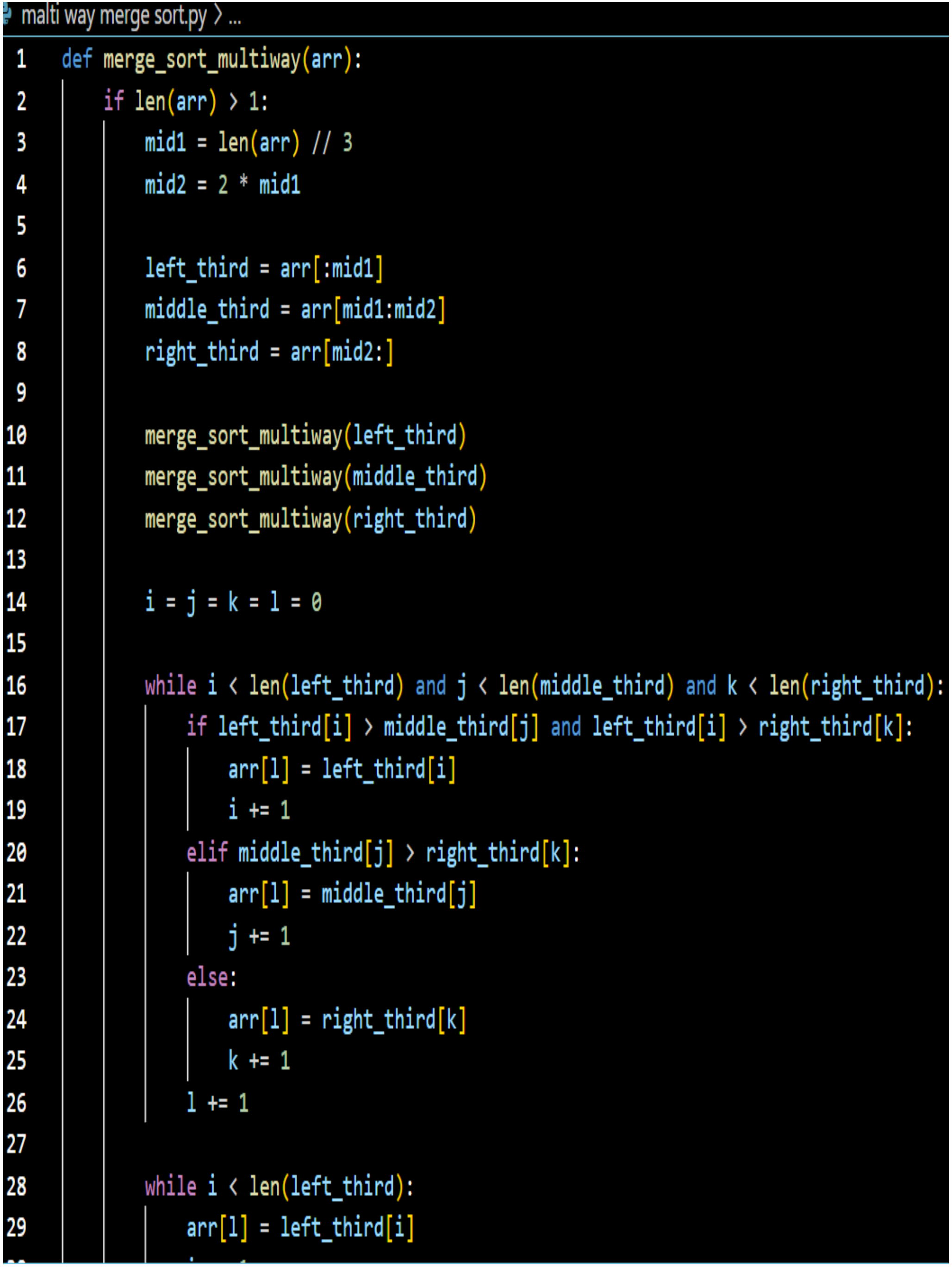
3. : Modify the merge sort algorithm to sort a list in descending order instead of ascending order. This will require changes to the merging step.



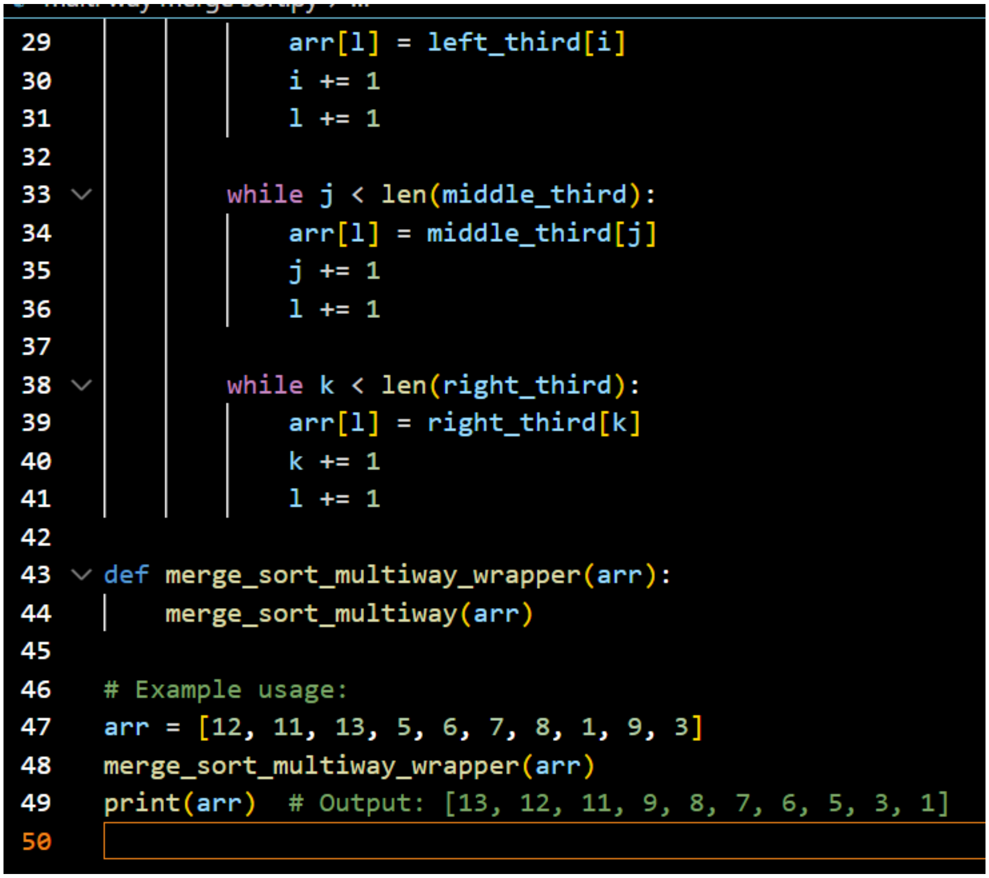
```python
def merge_sort_descending(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    left_half = merge_sort_descending(left_half)
    right_half = merge_sort_descending(right_half)

    return merge_descending(left_half, right_half)

def merge_descending(left, right):
    result = []
    i = 0
    j = 0

    while i < len(left) and j < len(right):
        if left[i] >= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])

    return result

arr = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
sorted_descending = merge_sort_descending(arr)
print(sorted_descending)
```

```
PS D:\2\3rd Semester\DSA\codes> & C:/Users/waqas/AppData/Loca
[9, 6, 5, 5, 5, 4, 3, 3, 2, 1, 1]
PS D:\2\3rd Semester\DSA\codes>
```

4. : Extend the merge sort algorithm to work with three or more sublists at each step, not just two. This is called a three-way (or multi-way) merge sort.



```
def merge_sort_multiway(arr):
    if len(arr) > 1:
        mid1 = len(arr) // 3
        mid2 = 2 * mid1

        left_third = arr[:mid1]
        middle_third = arr[mid1:mid2]
        right_third = arr[mid2:]

        merge_sort_multiway(left_third)
        merge_sort_multiway(middle_third)
        merge_sort_multiway(right_third)

        i = j = k = l = 0

        while i < len(left_third) and j < len(middle_third) and k < len(right_third):
            if left_third[i] > middle_third[j] and left_third[i] > right_third[k]:
                arr[l] = left_third[i]
                i += 1
            elif middle_third[j] > right_third[k]:
                arr[l] = middle_third[j]
                j += 1
            else:
                arr[l] = right_third[k]
                k += 1
            l += 1

        while i < len(left_third):
            arr[l] = left_third[i]
```



```python
            arr[l] = left_third[i]
            i += 1
            l += 1

        while j < len(middle_third):
            arr[l] = middle_third[j]
            j += 1
            l += 1

        while k < len(right_third):
            arr[l] = right_third[k]
            k += 1
            l += 1

def merge_sort_multiway_wrapper(arr):
    merge_sort_multiway(arr)

# Example usage:
arr = [12, 11, 13, 5, 6, 7, 8, 1, 9, 3]
merge_sort_multiway_wrapper(arr)
print(arr)  # Output: [13, 12, 11, 9, 8, 7, 6, 5, 3, 1]

```